A Parallel Multigrid Preconditioned Conjugate Gradient Algorithm for Groundwater Flow Simulations

S. F. Ashby
R. D. Falgout
Center for Computational Sciences & Engineering

CIRCULATION COPY
SUBJECT TO RECALL
IN TWO WEEKS

Lawrence Livermore National Laboratory

ParFlow

LOAN COPY

UCRL-JC-122359
October 1995

THE PARFLOW PROJECT

The goal of the PARFLOW project is to apply high performance computing techniques to the three-dimensional modeling of fluid flow and chemical transport in heterogeneous porous media to enable more realistic simulations of subsurface contaminant migration. This is an interdisciplinary effort involving Laboratory scientists from the Center for Computational Sciences & Engineering, the Earth Sciences Division of Environmental Programs, and the Environmental Protection Department. Lawrence Livermore National Laboratory. International Technology Corporation, and Cray Research, Incorporated are partnering to commercialize some of the work described herein under the auspices of one or more Cooperative Research and Development Agreements (CRADAs).

Various aspects of this work were funded by one or more of the following: DOE Office of Scientific Computing (Applied Mathematical Sciences), DOE Defense Programs (Technology Transfer Initiative), and Laboratory Directed Research and Development.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

REPRINT

This is a reprint of a paper that has been submitted to a special issue of *Nuclear Science & Engineering* devoted to the 1995 ANS International Conference on Mathematics and Computations, Reactor Physics, and Environmental Analyses, held April 30-May 4, 1995, in Portland, OR. This preprint is made available with the understanding that it will not be cited or reproduced without the permission of the authors.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This report has been reproduced
directly from the best available copy.

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information
P.O. Box 62, Oak Ridge, TN 37831
Prices available from (615) 576-8401, FTS 626-8401

Available to the public from the
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.,
Springfield, VA 22161

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.

UCP1

BEST AVAILABLE COPY

FOR ORIGINAL REPORT

CALL

REPORTS LIBRARY

X37097

A PARALLEL MULTIGRID PRECONDITIONED CONJUGATE GRADIENT ALGORITHM FOR GROUNDWATER FLOW SIMULATIONS*

STEVEN F. ASHBY† AND ROBERT D. FALGOUT‡

Abstract. This paper discusses the numerical simulation of groundwater flow through heterogeneous porous media. The focus is on the performance of a parallel multigrid preconditioner for accelerating convergence of conjugate gradients, which is used to compute the pressure head. The numerical investigation considers the effects of boundary conditions, coarse grid solver strategy, increasing the grid resolution, enlarging the domain, and varying the geostatistical parameters used to define the subsurface realization. Scalability is also examined. The results were obtained using the PARFLOW groundwater flow simulator on the CRAY T3D massively parallel computer.

Key words. multigrid, conjugate gradients, preconditioning, groundwater

AMS(MOS) subject classifications. 65M55, 65F10, 65Y10, 76S05

1. Motivation. The numerical simulation of subsurface fluid flow and chemical migration plays an increasingly important role in several environmental applications, including groundwater remediation studies and groundwater resource management. Although sophisticated simulations have been used for decades in the petroleum industry with considerable success, they have been less widely used in environmental applications, but they are gaining in popularity as sites become larger and more complex. Computational environmental remediation is particularly attractive for the design, evaluation, and management of engineered remediation procedures [24], especially for large industrial and government sites. Simulations can be used, for instance, to choose the best cleanup strategy for a given site, and then, once a scheme is chosen, to manage it in the most cost effective fashion. They also can be used to perform more realistic risk assessment in support of key decision-making, as well as an aid in demonstrating regulatory compliance.

Mathematically, the key to such simulations is the solution of the large, sparse system of linear equations resulting from the discretization of a second order elliptic partial differential equation with a widely varying coefficient function. The solution of this system yields the subsurface fluid flow velocities, which are then used to track groundwater flow and contaminant migration. In this paper, we introduce a multigrid preconditioned conjugate gradient algorithm for solving these systems, and we investigate its performance on a variety of realistic problems. Since we are interested in detailed simulations with millions of spatial zones, we employ massively parallel processing power. In particular, we will describe the parallel implementation and performance of our algorithm on the CRAY T3D computer.

1.1. The need for improved modeling. Many of the computer codes in use today make unrealistic assumptions about the nature of the subsurface medium and the associated flow behavior. For example, many codes assume that the subsurface is homogeneous in composition and spatial distribution, and ignore altogether variations in the vertical dimension. As a result,

---

* This work was supported by Laboratory Directed Research and Development, and by the Applied Mathematical Sciences program of the Office of Scientific Computing, Department of Energy, by Lawrence Livermore National Laboratory under contract W-7405-ENG-48.

† Center for Computational Sciences & Engineering, Lawrence Livermore National Laboratory, P.O. Box 808, L-316, Livermore, CA 94551 (sfashby@llnl.gov).

‡ Center for Computational Sciences & Engineering, Lawrence Livermore National Laboratory, P.O. Box 808, L-316, Livermore, CA 94551 (rfalgout@llnl.gov).

these codes may fail to represent accurately many important processes. Consequently, the conclusions drawn from simulations made with these codes are open to question, as are the decisions based on these conclusions.

In reality, the subsurface is three-dimensional and heterogeneous. This means that some regions of the subsurface are more permeable to water flow than others; this is represented by a spatially variable flow parameter known as the *hydraulic conductivity*. The heterogeneous nature of the subsurface gives rise to *preferential flow channels* in the subsurface velocity field, which can have a dramatic impact on fluid flow and contaminant transport [1, 3]. For example, these channels can lead to "fingering" in contaminant migration, that is, nonuniform dispersion over time. It is essential to resolve this behavior because it can drastically alter the conclusions one makes about a given remediation procedure. For example, a homogeneous model may yield simulations that predict that the procedure under study will meet regulatory requirements. However, a more accurate heterogeneous model (with adequate resolution) may predict the opposite. Regulatory agencies are now recognizing this and demanding the increased use of detailed, three-dimensional modeling.

1.2. The role of high performance computing. Researchers have recognized the deficiencies of the simplified homogeneous codes for some time, but have been unable to consider running more realistic simulations until recently. Current simulations often lack sufficient spatial resolution (to capture fingering) because of a paucity of subsurface data and the inability to solve the resulting problems on even the largest of conventional vector supercomputers.

The size of the site to be modeled (typically several square kilometers) and the need to resolve these heterogeneities adequately (on the order of meters), leads to computational domains with upwards of one billion spatial zones. The use of adaptive gridding and local refinement can reduce the total number of zones needed by one or two orders of magnitude, but one is still left with huge problems that quickly overwhelm all but the largest of conventional supercomputers. Moreover, we need to run hundreds of such simulations as we conduct time-dependent studies, examine different remediation or production strategies, or run the code in a Monte Carlo fashion or within an optimization code. In light of these considerations, it is necessary to employ massively parallel processing power, and toward this end, we are building a parallel flow simulator called PARFLOW. It is designed to be portable and scalable across a variety of distributed memory MIMD machines with message passing, ranging from workstation clusters to large MPPs.

Massively parallel processing may be necessary for detailed simulations, but it is not sufficient. One also needs to employ high performance *algorithms*, that is, accurate and fast numerical techniques that can be implemented efficiently on these machines. As we will see, simply changing the linear solver can result in two orders of magnitude reduction in CPU time. This is especially important in time-dependent simulations, where the right numerical method can mean the difference between a 30-*hour* run and a 30-*minute* run on an MPP.

1.3. Overview of paper. In this paper, we will investigate the performance of a parallel multigrid preconditioner for accelerating convergence of conjugate gradients, which is used to compute a pressure quantity. Our numerical investigation considers the effects of boundary conditions, coarse grid solver strategy, increasing the grid resolution, enlarging the domain, and varying the geostatistical parameters used to define the subsurface realization. Scalability is also examined. The results were obtained using the PARFLOW groundwater flow simulator on the CRAY T3D massively parallel computer.

The paper is organized as follows: We present our mathematical model and numerical discretization in §2. The multigrid preconditioned conjugate gradient algorithm is described in detail in §3, and its implementation is discussed in §4. The results of our numerical investigation and parallel performance study are given in §5.

2. Numerical simulation of groundwater flow. We consider steady state saturated (i.e., single phase) flow, which is of practical interest because contaminant transport is most rapid in this region. It is essential that we be able to solve such problems quickly and accurately because a similar elliptic problem will constitute the main computational cost of the multiphase, time-dependent simulations in which we are ultimately interested. In particular, we need an efficient and scalable elliptic solver, which we have found in the multigrid preconditioned conjugate gradient algorithm described in this paper.

Our mathematical model of groundwater flow is based on Darcy's law and conservation of mass in a porous medium, which may be combined and rewritten as

$$-\nabla\cdot(K\nabla(h+z)) - Q = 0 \tag{2.1}$$

where h is the *pressure head*, K is the hydraulic conductivity (i.e., problem specification), and Q is a source term (used to represent pumping wells, for example). At present, the problem domain is assumed to be a parallelepiped; the boundary conditions may be Dirichlet or flux.

The hydraulic conductivity realization is central to the problem definition; it is embodied in the K function in equation (2.1). Of course, one never has enough data (i.e., direct measurements) to characterize a given site completely (i.e., to completely specify K). To develop the detailed subsurface characterization needed for the type of simulation described above, hydrogeologists typically employ geostatistical techniques to create statistically accurate *realizations* of key subsurface properties, particularly the hydraulic conductivity [23]. Monte Carlo and optimization techniques can be used to quantify the inherent uncertainty and enable site managers to perform more realistic risk assessments. Although these realizations cannot give the precise value of the hydraulic conductivity at an (x, y, z) coordinate, they do reproduce the *statistical* patterns of heterogeneity observed in real systems, and can be used to evaluate various remediation strategies, say, determining the optimal pumping configuration in a pump-and-treat scheme.

We use Tompson's turning bands algorithm [23] to generate K. This is a technique for computing a spectral random field with given statistical properties. Specifically, one specifies a geometric mean μ for the K field, a variance σ^2 for the $\ln(K)$ field, and correlation lengths λ_x, λ_y, and λ_z. See [23] for a description of the turning bands algorithm, and see [6] for a discussion of its parallel implementation.

The heterogeneous nature of the subsurface manifests itself in the variability of K, that is, in the variability of the coefficient function for the elliptic PDE. In practice, this coefficient function may vary by as many as ten orders of magnitude, and so the function is effectively discontinuous. This results in an ill-conditioned linear system. In designing our multigrid algorithm, we must be careful to consider the discontinuous nature of the function when defining the interpolation and restriction operators, as well as the coarse grid operator.

2.1. Discrete solution approach. We employ a standard 7-point finite volume spatial discretization on a uniform mesh. After discretization, we obtain a large system of linear equations, $Ah = f$. The coefficient matrix A is symmetric positive definite and has the usual seven stripe pattern. The matrix has order $N = n_x \times n_y \times n_z$, where the n_i are the number of grid points in the x, y, and z directions, respectively. For problems of interest, N is in the millions; the large number is dictated by the size of the physical site and the need to resolve heterogeneities adequately. Once the pressure head is computed, the velocity field can be calculated easily using a simple differencing scheme. This field is then passed to a transport code to simulate contaminant migration.

The solution of the large linear system is computationally intensive and must be done efficiently and accurately. Since we are interested in detailed simulations (i.e., high resolution), we must use an iterative scheme. Within the hydrology community, the most commonly used methods are SIP and SSOR. Recently, however, the more powerful conjugate gradient method (CGHS) [17], and its preconditioned version (PCG) [10], have been used with great success. For example, polynomial preconditioned conjugate gradients was shown [19] to be an order of magnitude faster than SIP and SSOR.

Multigrid algorithms also are attractive for these types of problems. These techniques are among the fastest currently available for the solution of linear systems arising from the discretization of elliptic partial differential equations. Unlike most other iterative methods, a good multigrid solver's rate of convergence is independent of problem size, meaning that the number of iterations remains fairly constant. Hence, both the multigrid algorithm and its parallel implementation are highly scalable (see §4.1). On the other hand, multigrid (MG) algorithms tend to be problem specific and less robust than Krylov iterative methods such as conjugate gradients (CG). Fortunately, it is easy to combine the best features of MG and CG into one algorithm: multigrid preconditioned conjugate gradients (MGCG). The resulting algorithm is robust, efficient, and scalable. Another advantage of this approach is that one can quickly implement a simple multigrid algorithm that is extremely effective as a preconditioner, but perhaps less effective as a stand-alone solver. This is especially valuable when the underlying PDE has a nearly discontinuous coefficient function, as in our case.

In this paper, we present results for multigrid and 2-step Jacobi preconditionings. Our emphasis will be on the multigrid preconditioner described in the next section. The 2-step Jacobi preconditioner is implemented via an inner iteration consisting of two steps of the basic Jacobi method; see, e.g., [15, pages 384-385] for details.

3. The MGCG algorithm.

In this section, we define our multigrid preconditioned conjugate gradient algorithm, MGCG. We first describe the MG preconditioner, the key components of which are discussed in each of the following sections. These include: the coarsening strategy; the prolongation and restriction operators, P and R; the coarse grid operator, A^c; the smoother, S; and the coarsest grid solver.

The 2-level MG algorithm is defined as follows:

```
        for $i = 0, 1, \ldots$ until convergence:
(3.1a)      $h^- = S(h_i, A, f, m)$
(3.1b)      $r^c = R(f - Ah^-)$
(3.1c)      $e^c = (A^c)^{-1} r^c$
(3.1d)      $h^+ = h^- + Pe^c$
(3.1e)      $h_{i+1} = S(h^+, A, f, m)$
        end for
```

In (3.1a) we perform m smoothing steps on the fine system of equations (we choose $m = 1$ in this paper). We then restrict the residual to the coarse grid in (3.1b). In step (3.1c) we solve the coarse system of equations, yielding a coarse grid approximation to the fine grid error. This coarse grid error is then prolonged (i.e., interpolated) to the fine grid, and added to the current fine grid solution approximation in step (3.1d). Finally, in (3.1e), we carry out m more smoothing steps on the fine system of equations. Steps (3.1b)–(3.1d) together are called the *correction step*, and the above algorithm describes a 2-level multigrid *V-cycle*. The full multilevel algorithm is defined by recursively applying the 2-level method to the system of equations in (3.1c). In other words, instead of solving (3.1c) exactly, we obtain an approximate solution by applying one V-cycle of the 2-level algorithm. This yields a new, coarser system of equations, which we may also solve approximately by applying the 2-level algorithm. This process is continued until we reach some *coarsest* system of equations, which is then solved to complete the V-cycle.

Before we continue, we need to introduce some notation. The fine grid matrix A has the following *stencil structure*:

$$A = \begin{bmatrix} -a^L_{i,j,k} \end{bmatrix} \begin{bmatrix} & -a^N_{i,j,k} & \\ -a^W_{i,j,k} & a^C_{i,j,k} & -a^E_{i,j,k} \\ & -a^S_{i,j,k} & \end{bmatrix} \begin{bmatrix} -a^U_{i,j,k} \end{bmatrix} \tag{3.2}$$

where W, E, S, N, L, U, and C are used mnemonically to stand for west, east, south, north, lower, upper, and center, respectively. Now, split A such that

$$A = T + B \tag{3.3}$$

where

$$T = \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} & 0 & \\ -a^W_{i,j,k} & t_{i,j,k} & -a^E_{i,j,k} \\ & 0 & \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \tag{3.4}$$

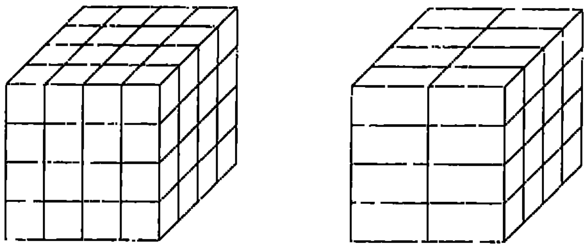
$$B = \begin{bmatrix} -a^L_{i,j,k} \end{bmatrix} \begin{bmatrix} & -a^N_{i,j,k} & \\ 0 & b_{i,j,k} & 0 \\ & -a^S_{i,j,k} & \end{bmatrix} \begin{bmatrix} -a^U_{i,j,k} \end{bmatrix} \tag{3.5}$$

and where

$$\begin{aligned} t_{i,j,k} &= a^C_{i,j,k} - b_{i,j,k} \\ b_{i,j,k} &= a^S_{i,j,k} + a^N_{i,j,k} + a^L_{i,j,k} + a^U_{i,j,k}. \end{aligned}$$

Note that A is split in the x direction: T contains the off-diagonal coefficients of A corresponding to the x direction and B describes the coupling in the y and z directions. We similarly split A in the y and z directions, but for clarity, we will use only the above x splitting in the discourse that follows.

3.1. Heuristic semi-coarsening strategy. Because the ground subsurface is generally stratified in nature, our computational grids typically have skewed cell aspect ratios. This produces anisotropy in the problem which causes "standard" MG algorithms to converge slowly. To ameliorate this problem, we employ a *semi-coarsening* strategy in which the grid is coarsened



fine grid　　　　coarse grid

FIG. 3.1. *Semi-coarsening in the x direction.*

in one spatial direction at a time. Semi-coarsening in the x direction is illustrated in Figure 3.1; the coarse grid is defined by taking every other yz plane.

To determine the direction of semi-coarsening, we use a heuristic based on the grid spacing. The algorithm chooses a direction with smallest spacing (i.e., strongest coupling). If this minimum spacing occurs in one or more directions, the algorithm attempts to coarsen first in x, then in y, and finally in z. One important issue in this scheme is determining how and when to terminate the coarsening algorithm. As we will see in §5, these issues can have a dramatic impact on the performance of the MG algorithm. The results presented there indicate that, in our MG algorithm, semi-coarsening down to a $1 \times 1 \times 1$ grid is optimal for typical groundwater problems.

In our numerical experiments, we show that this semi-coarsening strategy effectively ameliorates anisotropies due to large grid cell aspect ratios. However, it does not take into account anisotropies in the rock matrix (i.e., the permeability tensor). We are currently investigating this issue, especially the relevant work discussed in [13, 16, 20, 21].

3.2. Operator-induced prolongation and restriction. One of the keys to a successful multigrid algorithm is the definition of the prolongation operator, P, which defines how vectors on a coarse grid are mapped onto the next finer grid. In the case of constant coefficient elliptic PDEs, P is usually defined via a simple interpolation scheme. However, when the coefficient function varies greatly, as in our problem, this is inadequate. Instead, one should use *operator-induced* prolongation, meaning that P is defined in terms of the coefficients of the fine grid matrix. Our prolongation operator is similar to those described in [2, 11, 14].

To elucidate, consider the prolongation of an error vector, e^c, from the coarse grid, $\mathcal{G}^c$, to the fine grid, $\mathcal{G}$. For the sake of discussion, let us assume that $\mathcal{G}^c$ is obtained by coarsening $\mathcal{G}$ in the x direction, as in Figure 3.1. (To be precise, we actually have prolongation operators P_x, P_y, and P_z, corresponding to each of the directions of semi-coarsening, but we will drop the subscripts for clarity below.) Prolongation is then defined by

$$P e^c = \begin{cases} p^W_{i,j,k} e^c_{i-1,j,k} + p^E_{i,j,k} e^c_{i+1,j,k}, & x_{i,j,k} \in \mathcal{G} \setminus \mathcal{G}^c \\ e^c_{i,j,k}, & x_{i,j,k} \in \mathcal{G}^c \end{cases} \tag{3.6}$$

where

$$\begin{aligned} p^W_{i,j,k} &= a^W_{i,j,k}/t_{i,j,k} \\ p^E_{i,j,k} &= a^E_{i,j,k}/t_{i,j,k}. \end{aligned}$$

In other words, at points on the fine grid that *are not* also on the coarse grid, the value of the prolonged error vector is defined as a weighted average of x-adjacent coarse grid error components. At points on the fine grid that *are* also on the coarse grid, the value of the prolonged error vector is the same as the corresponding coarse grid error component. Prolongation in y and z is defined analogously.

The restriction operator, R, is used to project from a fine grid to a coarse grid. As is commonly done, we define $R = P^T$.

3.3. Definition of coarse grid operator. Another important issue in multigrid is the definition of the coarse grid operator, A^c. In the literature, this matrix is often taken to be the *Galerkin matrix*, $P^T AP$. This choice for A^c is optimal in the sense that the quantity $\|e+Pe^c\|_A$ is minimized over all coarse grid vectors e^c. That is, after a multigrid correction step, the error in the approximate solution is minimized in the A-norm for a given choice of prolongation operator, P. In particular, if the error before correction is in the range of prolongation, then the correction step yields the *exact* solution. The drawback of this coarse grid operator is that it has a 27-point stencil, which requires additional storage and does not allow us to define the multi-level algorithm by recursively applying the 2-level algorithm.

Another way to define A^c is to re-discretize the differential equation on the coarse grid. This has the benefit of yielding a 7-point stencil structure, which requires less storage than the 27-point stencil, and allows recursive definition of a multi-level algorithm. On the other hand, this operator lacks the minimization property of the Galerkin operator.

In our algorithm, we attempt to combine the best of both approaches by algebraically defining A^c as (again assuming semi-coarsening in the x direction)

$$A^c = T^c + B^c \tag{3.7}$$

where

$$T^c = P^T TP = \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} & 0 & \\ -a^{c,W}_{i,j,k} & t^c_{i,j,k} & -a^{c,E}_{i,j,k} \\ & 0 & \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \tag{3.8}$$

$$\begin{aligned} a^{c,W}_{i,j,k} &= a^W_{i,j,k} p^W_{i-1,j,k} \\ a^{c,E}_{i,j,k} &= a^E_{i,j,k} p^E_{i+1,j,k} \\ t^c_{i,j,k} &= t_{i,j,k} - a^W_{i,j,k} p^E_{i-1,j,k} - a^E_{i,j,k} p^W_{i+1,j,k} \end{aligned}$$

and

$$B^c = \begin{bmatrix} -a^{c,L}_{i,j,k} \end{bmatrix} \begin{bmatrix} & -a^{c,N}_{i,j,k} & \\ 0 & b^c_{i,j,k} & 0 \\ & -a^{c,S}_{i,j,k} & \end{bmatrix} \begin{bmatrix} -a^{c,U}_{i,j,k} \end{bmatrix} \tag{3.9}$$

$$\begin{aligned} a^{c,S}_{i,j,k} &= a^S_{i,j,k} + \frac{1}{2} a^S_{i-1,j,k} + \frac{1}{2} a^S_{i+1,j,k} \\ a^{c,N}_{i,j,k} &= a^N_{i,j,k} + \frac{1}{2} a^N_{i-1,j,k} + \frac{1}{2} a^N_{i+1,j,k} \\ a^{c,L}_{i,j,k} &= a^L_{i,j,k} + \frac{1}{2} a^L_{i-1,j,k} + \frac{1}{2} a^L_{i+1,j,k} \\ a^{c,U}_{i,j,k} &= a^U_{i,j,k} + \frac{1}{2} a^U_{i-1,j,k} + \frac{1}{2} a^U_{i+1,j,k} \\ b^c_{i,j,k} &= a^{c,S}_{i,j,k} + a^{c,N}_{i,j,k} + a^{c,L}_{i,j,k} + a^{c,U}_{i,j,k}. \end{aligned}$$

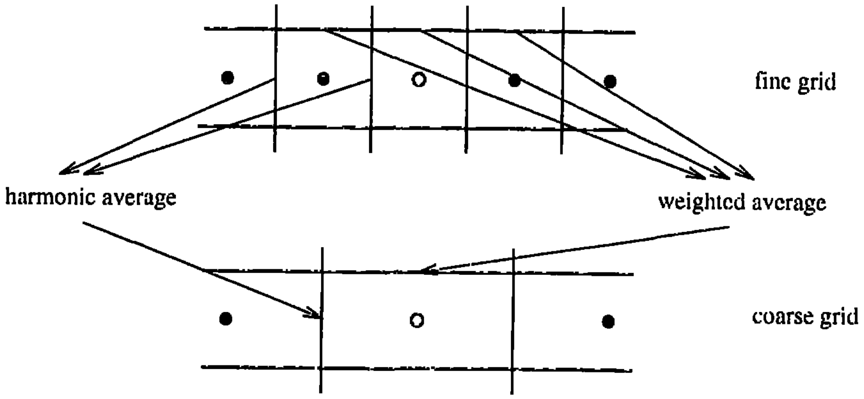


FIG. 3.2. *Definition of the coarse grid operator (2-d illustration with x semi-coarsening).*

In other words, A^c is a Galerkin operator in x (the direction of semi-coarsening) plus a weighted sum of y and z stencil coefficients. The coefficients in (3.9) describe the connections in y and z of the coarse grid variables, and our reason for choosing these particular weights is illustrated below.

Away from the domain boundaries, the algebraic definition of A^c in (3.7)–(3.9) also may be interpreted geometrically as the result of a finite volume discretization of (2.1) on the coarse grid $\mathcal{G}^c$. Consider the grid point marked "o" in Figure 3.2 (where we illustrate only two dimensions for simplicity). The finite volume discretization requires hydraulic conductivity values on the cell faces about this grid point. To generate the matrix coefficients, these values are first multiplied by the area of the cell face, then divided by the grid spacing in the perpendicular direction. So, if grid point "o" in Figure 3.2 has index (i, j, k), then on the fine grid we have

$$\begin{aligned} a^W_{i,j,k} &= \frac{\Delta y \Delta z}{\Delta x} K_{i-1/2,j,k} \\ a^E_{i,j,k} &= \frac{\Delta y \Delta z}{\Delta x} K_{i+1/2,j,k} \\ a^S_{i,j,k} &= \frac{\Delta x \Delta z}{\Delta y} K_{i,j-1/2,k} \\ a^N_{i,j,k} &= \frac{\Delta x \Delta z}{\Delta y} K_{i,j+1/2,k}. \end{aligned} \tag{3.10}$$

Now, consider the finite volume discretization on the coarse grid. We first compute hydraulic conductivity values on coarse-grid cell faces as in Figure 3.2; for vertical faces, we take a harmonic average of values on adjacent fine-grid cell faces, and for horizontal cell faces, we take an arithmetic average of values on corresponding fine-grid cell faces. Since the x grid spacing on $\mathcal{G}^c$ is twice that on $\mathcal{G}$, we have that

$$\begin{aligned} a^{c,W}_{i,j,k} &= \frac{\Delta y \Delta z}{2\Delta x} \left(\frac{2K_{i-3/2,j,k} K_{i-1/2,j,k}}{K_{i-3/2,j,k} + K_{i-1/2,j,k}} \right) \\ a^{c,E}_{i,j,k} &= \frac{\Delta y \Delta z}{2\Delta x} \left(\frac{2K_{i+3/2,j,k} K_{i+1/2,j,k}}{K_{i+3/2,j,k} + K_{i+1/2,j,k}} \right) \\ a^{c,S}_{i,j,k} &= \frac{2\Delta x \Delta z}{\Delta y} \left(\frac{1}{2} K_{i,j-1/2,k} + \frac{1}{4} K_{i-1,j-1/2,k} + \frac{1}{4} K_{i+1,j-1/2,k} \right) \\ a^{c,N}_{i,j,k} &= \frac{2\Delta x \Delta z}{\Delta y} \left(\frac{1}{2} K_{i,j+1/2,k} + \frac{1}{4} K_{i-1,j+1/2,k} + \frac{1}{4} K_{i+1,j+1/2,k} \right). \end{aligned} \tag{3.11}$$

Using (3.10) in (3.11), and noting that

$$t_{i,j,k} = a^W_{i,j,k} + a^E_{i,j,k}$$

holds in the middle of the domain, it is easy to see that the coefficients produced by this finite volume discretization on the coarse grid are the same as those given in (3.7)–(3.9).

3.4. Smoothers. The smoother is another important part of a multigrid algorithm. A "good" smoother complements the correction step by damping modes that the correction step does not. However, as is often the case with numerical algorithms, the smoother that does the best job of damping these errors is typically the most computationally expensive. For example, line and plane methods are generally better than are pointwise methods at damping high frequency error components, but they are computationally more expensive and less parallelizable.

We use simple pointwise *damped Jacobi* (with weighting factor 2/3) and *red/black Gauss-Seidel* smoothers in our MG algorithm. Although these smoothers are easy to implement scalably in parallel, the resulting MG algorithms lack robustness. However, as we will see (§3.6), we regain robustness by using MG as a preconditioner within a CG algorithm—without the additional coding complexity (and possibly greater overhead) of a line or plane smoothing.

3.5. Coarsest grid solvers. To complete the multigrid algorithm, we must decide when to stop the coarsening procedure, and how to solve the coarsest system of equations. For example, should we solve the coarsest system of equations exactly, or just do a few smoothing steps to obtain an approximate solution? In §5.1, we run several experiments in this regard, and we conclude that coarsening down to a 1 × 1 × 1 grid is optimal for our algorithm and for this application. The 1 × 1 × 1 "system" is solved exactly via one sweep of red-black Gauss-Seidel. (We employ CGHS and red/black Gauss-Seidel as our coarse grid solvers. One could also consider a direct solution of the coarsest grid system via Gaussian elimination, but the iterative solvers are adequate for our purposes.)

3.6. Stand-alone multigrid versus multigrid as a preconditioner. Although multigrid (MG) algorithms are extremely fast, they tend to be problem-specific and less robust than Krylov iterative methods such as conjugate gradients (CG). Fortunately, it is easy to combine the best features of MG and CG into a *multigrid preconditioned conjugate gradient* (MGCG) algorithm that is robust, efficient, and scalable. The main advantage of this approach is that one can quickly implement a simple multigrid algorithm that is extremely effective as a preconditioner, but perhaps less effective as a stand-alone solver.

The well-known PCG method (*Orthomin* implementation) [7, 10] is given by

```
(3.12a)    $p_0 = s_0 = Cr_0$
           for $i = 0, 1, \ldots$ until convergence:
(3.12b)        $\alpha_i = \frac{\langle r_i, s_i \rangle}{\langle Ap_i, p_i \rangle}$
(3.12c)        $x_{i+1} = x_i + \alpha_i p_i$
(3.12d)        $r_{i+1} = r_i - \alpha_i A p_i$
(3.12e)        $s_{i+1} = C r_{i+1}$
(3.12f)        $\beta_i = \frac{\langle r_{i+1}, s_{i+1} \rangle}{\langle r_i, s_i \rangle}$
(3.12g)        $p_{i+1} = s_{i+1} + \beta_i p_i$
           end for
```

In the MGCG algorithm, the preconditioning operator, C, is never explicitly formed. Instead, (3.12e) is effected by applying the MG algorithm (3.1) to the residual system of equations, $Ae = r$, using an initial guess of $e_0 = \vec{0}$. The resulting approximate solution is s_{i+1}.

When designing a preconditioner for CG, one needs to insure that the preconditioning matrix is symmetric, and preferably positive definite. For multigrid preconditioning, this condition is satisfied by doing an equal number of symmetric smoothing steps both before and after each coarse grid correction. (The smoothing step is symmetric if the iteration matrix of the associated method is symmetric.) However, this is not necessarily required (see, e.g., [22]). Multigrid algorithms also can be applied to nonsymmetric problems (e.g., [12]) and to problems with irregular meshes (e.g., [18]).

Our current implementation of MGCG is simple but effective. The MG preconditioning step consists of a single *V-cycle* (as defined above) with a choice of weighted Jacobi or symmetric red/black Gauss-Seidel smoothing. We use an equal number, m, of smoothing steps before and after correction. (In this paper, $m = 1$.)

4. Parallel implementation.

The MGCG algorithm described above has been implemented in ParFlow, a portable and scalable parallel flow simulator. The algorithms in ParFlow all employ a straightforward data decomposition approach to parallelism. Specifically, problem data is distributed across a logical three-dimensional process grid topology consisting of $P = p \times q \times r$ processes. The data within a process is viewed as a three-dimensional *subgrid* of grid points (as defined by the discretization of equation (2.1)). Vector data owned by a process is called a *subvector*, and each element of a subvector is associated with a grid point in the process' subgrid. Similarly, matrix data owned by a process forms a *submatrix*. The rows of this submatrix are viewed as stencils, and each stencil is associated with a grid point in the process' subgrid. Note that although we distribute the problem data by decomposing the problem domain, we are *not* doing domain decomposition in the algorithmic sense. We are solving the *full* problem rather than independent subproblems.

Computations in ParFlow proceed in an *owner computes* fashion. That is, processes only do computations associated with their local subgrid, taking care to exchange data with neighboring processes when needed. For example, consider matrix-vector multiplication, $y = Ax$, a key operation in the CG solver. To compute the matvec result at a given grid point (i, j, k), we "apply" the stencil to the grid: For each neighboring grid point specified by the stencil, we multiply the vector value at that point by the corresponding stencil coefficient, and then sum these products. (This is equivalent to multiplying a row of the matrix A by the vector x.) However, at subgrid boundary points, some stencil coefficients may reach outside of the process' subgrid. At these points, we must first communicate data from neighboring processes. In general, these communications patterns can be quite complicated. Take, for example, pointwise red/black Gauss-Seidel. Before a process can do a red sweep, it must exchange black boundary data with neighboring processes. Likewise, red boundary data must be exchanged before a black sweep can be completed. In order to simplify coding and speed application development, subvectors and submatrices have an additional layer of space set aside for storing this communicated boundary data. The grid points associated with this layer are called *ghost points*.

When possible, the communications and computations in ParFlow are scheduled so that they overlap. For example, in our matvec operation —for which the matrix A has a standard 7-point stencil—the computations away from subgrid boundaries can be done independently of the boundary data communications. If the parallel machine has the appropriate hardware support, we can do these computations and communications simultaneously. On most machines, and for most large problems, the communications will finish before the internal computations have completed, effectively masking the communications costs. Unfortunately, the CRAY T3D does not provide this support.

We remark that PARFLOW was designed to be parallel from its inception. In particular, computations are organized so as to avoid explicit data redistribution, thereby improving the code's efficiency and parallel performance. The choice of process grid topology P can have a significant impact on performance, largely due to cache issues [6]. Thus, in choosing the topology P, one must weigh the competing needs of various portions of the code and determine the best overall topology. One should *not* choose different topologies for different stages of the calculations and redistribute data within the simulation.

4.1. Scalability. One of the attractive features of multigrid is that it *can be* a scalable *algorithm*, meaning that the number of iterations required for convergence remains roughly constant as the grid is refined. This is true for both stand-alone multigrid and multigrid preconditioned conjugate gradients. The emphasis on *can be* is meant to stress the importance of properly defining the various key ingredients, especially the prolongation/restriction operators and the coarse grid operator. However, having a scalable algorithm is only half the equation: one also must have a scalable *parallel implementation.* Since we semi-coarsen to a $1 \times 1 \times 1$ problem, it is impossible for us to have a truly scalable parallel implementation since the number of semi-coarsenings increases with the size of the problem (in a logarithmic fashion). Although the amount of work and the amount of communicated data *per processor* remains the same, the number of communication *calls* increases as the problem size increases (since communications are needed at each grid level in the V-cycle). This increased communication overhead is the only impediment to perfect scalability of the MG algorithm. For all practical purposes, however, both our algorithm (number of iterations) and its implementation (CPU time) are scalable, as we will show in §5.

Since we semi-coarsen the fine grid to a $1 \times 1 \times 1$ problem, a rather severe load imbalance results from the infamous "idle processor problem". However, this is not as serious as it might at first seem. In simulating groundwater flow through heterogeneous porous media for large sites, one must use a large number of spatial zones—often in the tens of millions. For such large problems, comparatively little work is being done while processes are idle (usually processes are idle at only a few of the coarsest grid levels). Hence, the effects of this inefficient use of resources is usually negligible. See, for example, [8, 9].

Finally, we comment that the CG part of the MGCG algorithm also is not perfectly scalable. This is because process data must be globally summed in order to compute the inner products needed in the algorithm. The communications required to do this grow logarithmically with the number of processes, but the effects of this increased communications is negligible (§5.5).

4.2. Portability via message-passing. We have successfully run PARFLOW (in various incarnations) on the following platforms: a single Sparcstation, a cluster of Sparcstations, a multiprocessor SGI Onyx, an nCUBE/2, an IBM SP-1, and the CRAY T3D. Portability is realized via message-passing. All message-passing primitives are localized within a machine-dependent library called AMPS, which has been layered on top of several message-passing systems, including the Reactive Kernel, PVM, Chameleon, and CRAY SHMEM calls. An MPI implementation is planned.

5. Numerical Results. In this section, we will investigate the performance of our multigrid algorithm in several contexts. In particular, we will study the effect of the following on the rate of convergence: (i) choice of boundary conditions, coarsest grid size, and coarsest grid

TABLE 5.1

Coarse grid solution strategy: no-flow boundary conditions on the top and bottom; Dirichlet ($H = 1$) conditions on the four vertical faces.

Coarse Grid Solver	MG iters	MG time	MGCG iters	MGCG time
1 step of RB on $3 \times 3 \times 3$	111	18.6	21	3.8
CGHS on $3 \times 3 \times 3$	58	10.9	19	3.9
CGHS on $5 \times 5 \times 3$	22	4.9	14	3.5
1 step of RB on $1 \times 1 \times 1$	19	3.5	10	2.1

solver; (ii) increasing the resolution (fixed domain size); (iii) enlarging the domain size (fixed grid spacing); and (iv) increasing the degree of subsurface heterogeneity. We also will describe the algorithm's parallel performance on the CRAY T3D massively parallel computer.

All of the experiments in this section are of the following form: The domain, $\Omega = L_x \times L_y \times L_z$, is a parallelepiped, where L_x, L_y, and L_z represent the domain lengths (in meters) in the x, y, and z directions, respectively. The grid is Cartesian with $N = n_x \times n_y \times n_z$ points and $\Delta = \Delta_x \times \Delta_y \times \Delta_z$ spacing. The subsurface is assumed to be a single, heterogeneous hydrostratigraphic unit with variable hydraulic conductivity K. To generate K, we use a turning bands algorithm [23] with geostatistical parameters μ, σ, λ_x, λ_y, and λ_z. Here $\ln \mu$ and σ represent the mean and standard deviation of the $\ln K$ field (μ also may be thought of as the geometric mean of K), and λ_x, λ_y, and λ_z represent the correlation lengths in the x, y, and z directions, respectively. Unless otherwise stated, we impose Dirichlet boundary conditions (*hydraulic head*, $H = h + z$, equal 1) on the four vertical sides of the domain, and no flow conditions on the top and bottom.

We consider three multigrid algorithms for solving the symmetric positive definite system of linear equations that results from the discretization of the elliptic pressure equation. Specifically, we compare the following: stand-alone multigrid with symmetric pointwise red/black Gauss-Seidel smoothing (MG); MG as a preconditioner for the conjugate gradient method (MGCG); and MGCG, but with damped Jacobi smoothing (MJCG). The preconditioning step consists of a single MG V-cycle. As discussed in the previous section, the smoothing operation should be implemented in a symmetric fashion when multigrid is used as a preconditioner in PCG. For comparison, we also consider PCG with 2-step Jacobi preconditioning (J2CG). Each of the algorithms was halted once the 2-norm of the relative residual was less than 10^{-9}. Unless otherwise noted, we used $P = 2 \times 4 \times 4$ processors of the CRAY T3D. (Some of the larger problems required a larger number of processors because of their memory needs.) All times are wall-clock times, and they are given in seconds. Although the test problems are contrived, they serve to illustrate the performance of the MGCG algorithm.

5.1. Effect of coarsest grid solver strategy. In Tables 5.1 and 5.2 we study the effect on convergence rate of the choice of coarsest grid solver strategy with respect to the type of boundary conditions. The experiment details are as follows:

$$\Omega = 1024 \times 1024 \times 25.6$$
$$N = 65 \times 65 \times 33, \quad \Delta = 16 \times 16 \times 0.8$$
$$\mu = 4, \quad \sigma = 1.5, \quad \lambda_x = 32, \quad \lambda_y = 32, \quad \lambda_z = 1.6$$

Let us consider first Table 5.1, in which we employ our "standard" boundary conditions: no flow on the top and bottom faces, and constant head ($H = 1$) on the remaining vertical faces. Let us also focus first on the issues related to multigrid. In row one, we coarsen to a

TABLE 5.2

Coarse grid solution strategy: Dirichlet ($H = 1$) boundary conditions on all faces.

	MG		MGCG	
Coarse Grid Solver	iters	time	iters	time
1 step of RB on $3 \times 3 \times 3$	12	2.0	8	1.6
CGHS on $3 \times 3 \times 3$	12	2.2	8	1.7
CGHS on $5 \times 5 \times 3$	12	2.1	8	1.7
1 step of RB on $1 \times 1 \times 1$	15	2.8	10	2.1

$3 \times 3 \times 3$ coarsest grid and then do one step of red/black GS. In row two, we again coarsen to a $3 \times 3 \times 3$ coarsest grid, but solve the coarsest system "exactly" via CGHS (which converged to machine tolerance in three iterations). This improves the convergence of MG considerably. The reason for this is that the system of equations on this coarsest grid is "almost singular" because of the strong coupling in the direction of a flux boundary condition (i.e., the z direction). Consequently, errors with "smooth" z components are not damped well by one step of GS smoothing.

In row three, we coarsen as in row two, except that we stop at a $5 \times 5 \times 3$ coarsest grid. We again use CGHS to solve this coarsest system; this time, it took 34-36 iterations to solve the coarsest grid problems to the specified tolerance (relative residual less than 10^{-9}). We see further significant improvement in convergence. To explain this, consider coarsening the $5 \times 5 \times 3$ grid first in x, and then in y, to a $3 \times 3 \times 3$ grid (as in the algorithm of row two). In each of these coarsening steps, we are coarsening in a direction orthogonal to the direction with strongest coupling (i.e., the z direction). These non-optimal coarsening steps actually slow convergence.

In row four, we coarsen to one equation in one unknown and solve it exactly via one step of red/black GS. This is the best method for this problem. Here, the heuristic semi-coarsening strategy coarsens in z in the "optimal" way until the z direction is eliminated altogether (thereby eliminating anisotropy in the z direction). This results in a coarse grid operator that looks like a 2D Laplacian. The remainder of the V-cycle (which involves coarsening only in the x and y directions) gives a good approximation to the solution of this system. Hence, this multigrid algorithm performs quite well.

The MGCG algorithm performs similarly for each of the experiments, except that it is much faster. Note that there is only a slight difference in iteration count between rows one and two, unlike for stand-alone MG. This is an indication that the MG algorithm in row one is having trouble with just a few of the modes, which the outer CG loop in MGCG easily eliminates.

In Table 5.2, we repeat these experiments with constant head ($H = 1$) on all six faces. The results are entirely different. First, we observe much faster convergence in this set of all-Dirichlet experiments. This is largely due to the near-singularity of the coarse grid matrices in the previous table, as discussed earlier. For the all-Dirichlet problems, it can be shown that both red/black GS and CGHS will solve the $3 \times 3 \times 3$ coarse grid problem in just one iteration. Since red/black GS is cheaper than CGHS, it is faster, as observed in the table. It also can be shown that CGHS will solve the $5 \times 5 \times 3$ coarse grid problems in just nine iterations. Although the algorithm in row three takes a bit longer to solve the coarsest grid problems, there is less semi-coarsening than in rows one and two, and the overall algorithm is competitive. Second, we notice that the worst case in Table 5.2 (row four) is the best case in Table 5.1. Since our finite volume discretization is vertex-centered, the boundary condition equations are not coupled to the other matrix equations. This, combined with our algebraic definition of the

TABLE 5.3

Increasing the spatial resolution: the domain size is fixed while the number of grid points is increased.

Problem Size			J2CG		MJCG		MGCG		MG	
n_x	n_y	n_z	iters	time	iters	time	iters	time	iters	time
17	17	9	456	1.1	12	0.3	9	0.4	12	0.4
33	33	17	963	5.7	13	0.5	10	0.7	17	1.1
65	65	33	1895	57.1	15	1.9	10	2.1	18	3.4
129	129	65	3706	772.1	16	10.8	11	12.6	21	20.6
257	257	129	7391	*1549.5	NA		11	*12.9	23	*23.9

*These times are for 256 processors ($P = 4 \times 8 \times 8$)

prolongation operator, results in prolongation coefficients that are zero at grid points near Dirichlet boundaries. Hence, the $n_x \times n_y \times 3$ coarse grid obtains no effective correction from the coarser $n_x \times n_y \times 2$ grid, which slows convergence of the algorithm.

We remark that the mixed boundary conditions used in Table 5.1 are more likely to arise in practice, and so we prefer the coarsening strategy used in row four.

5.2. Increasing the Spatial Resolution. In Table 5.3, we study the effect on convergence rate of increasing the spatial resolution. Specifically, we increase the number of grid points used to resolve each correlation length, but keep the problem domain fixed. (We start with two grid points per correlation length and increase to 32 grid points per correlation length.) The experiment details are as follows:

$$\Omega = 1024 \times 1024 \times 25.6$$
$$\Delta = 1024.0/(n_x - 1) \times 1024.0/(n_y - 1) \times 25.6/(n_z - 1)$$
$$\mu = 4, \quad \sigma = 1.5, \quad \lambda_x = 128, \quad \lambda_y = 128, \quad \lambda_z = 6.4$$

We see that increasing the spatial resolution has a significant effect on the convergence rate of J2CG (as expected), but has little effect on the MG-based algorithms. Specifically, the J2CG iteration count doubles when the resolution doubles (i.e, problem size increases by 2^3), but MGCG converges in about ten iterations independent of resolution. As the resolution increases, J2CG becomes increasingly impractical, and one must use a multigrid approach. For example, in the $257 \times 257 \times 129$ case, J2CG takes about 120 times longer to converge than MGCG, and this multiplier would grow if we increased the problem size further. Note also, that although MJCG takes more iterations to converge than MGCG, it converges a little faster. This is due to two things: (i) Jacobi has less communication overhead and, in general, runs at a higher MFLOP rate than red/black GS; and (ii) in MJCG we do two smoothings per grid level, but in MGCG, we do three smoothings because of an extra half sweep that is done to insure symmetry. Note that the stand-alone MG algorithm is not as effective as MGCG because of problems with a few extraneous modes (as explained earlier).

We remark that if we did not semi-coarsen to a grid with only 1 grid point in the z direction, the iteration counts in the first few rows of the table would be higher. This is because the first semi-coarsening in an x or y direction would occur *not* because the coupling in these directions was strongest ("optimal" coarsening strategy), but as a result of having too few z points. As discussed in §5.1, this would have an adverse effect on convergence which would be more pronounced for the smaller problem sizes. See [4] for related experiments.

Remark: The overall slow convergence of J2CG results partly from anisotropy in the problem due to the skewed grid cell aspect ratio. This is a consequence of how the eigenvalues of A

TABLE 5.4

Enlarging the domain size: the grid spacing is fixed while the number of grid points is increased.

Problem Size			J2CG		MJCG		MGCG		MG	
n_x	n_y	n_z	iters	time	iters	time	iters	time	iters	time
17	17	9	453	1.1	11	0.3	9	0.4	12	0.4
33	33	17	957	5.7	13	0.5	10	0.7	14	0.9
65	65	33	1860	56.0	16	2.0	10	2.1	19	3.6
129	129	65	3665	763.4	18	12.1	11	12.6	21	20.6
257	257	129	6696	*1403.8	NA		13	*15.1	22	*22.8

*These times are for 256 processors ($P = 4 \times 8 \times 8$)

are distributed. When the grid cell aspect ratio is near 1:1:1, the eigenvalues are more tightly clustered in the middle of the spectrum, and the *effective* condition number is less than the *true* condition number. (Recall that the CG rate of convergence is governed by the effective condition number, not the true condition number, because CG is able to damp outlying eigenvalues quickly.) When the grid cells are skewed, the eigenvalues cluster near the endpoints of the spectrum, and the effective and true condition numbers are nearly identical. Moreover, the flux boundary conditions on the z faces result in a larger effective condition number than would Dirichlet conditions, reducing further the effectiveness of J2CG on this problem.

5.3. Enlarging the size of the domain. In Table 5.4, we study the effect on convergence rate of growing the domain size. In some remediation studies, one wishes to enlarge the initial site to encompass neighboring property. This might be necessary, for instance, if a contaminant were discovered to have migrated off-site. In such a scenario, the engineer might wish to use the same geostatistics and grid spacing, but enlarge the domain by increasing the number of spatial zones. In our experiments, we maintain a constant two grid points per correlation length. The experiment details are as follows:

$$\begin{aligned}
&\Omega = (n_x - 1)\Delta_x \times (n_y - 1)\Delta_y \times (n_z - 1)\Delta_z \\
&\Delta = 4 \times 4 \times 0.2 \\
&\mu = 4, \quad \sigma = 1.5, \quad \lambda_x = 8, \quad \lambda_y = 8, \quad \lambda_z = 0.4
\end{aligned}$$

The results in Table 5.4 are qualitatively and quantitatively similar to the results in Table 5.3. The minor differences in the two tables are due to the differing subsurface realizations (produced by turning bands) in the two experiments.

5.4. Increasing Degree of Heterogeneity. In Table 5.5, we study the effect on convergence rate of increasing the degree of heterogeneity. This heterogeneity is represented by the parameter σ described earlier. The experiment details are as follows:

$$\begin{aligned}
&\Omega = 1024 \times 1024 \times 25.6 \\
&N = 129 \times 129 \times 65, \quad \Delta = 8 \times 8 \times 0.4 \\
&\mu = 4, \quad \lambda_x = 16, \quad \lambda_y = 16, \quad \lambda_z = 0.8
\end{aligned}$$

The first row in the table ($\sigma = 0.0$) corresponds to a homogeneous subsurface medium, in which case the coefficient function K is constant, and so the matrix A is Laplacian-like. As σ increases, so does the degree of heterogeneity. Specifically, the variance, σ_K^2, of the lognormally distributed conductivity field K increases exponentially. This variability in K causes the coefficient matrix A to becomes increasingly ill-conditioned. The effect on MG of

TABLE 5.5
Varying the degree of heterogeneity.

Heterogeneity		J2CG		MGCG		MG	
σ	σ_K^2	iters	time	iters	time	iters	time
0.0	0×10^0	1701	354.4	9	10.4	13	12.8
0.5	6×10^0	3121	650.3	9	10.4	13	12.8
1.0	7×10^1	3388	705.7	9	10.4	12	11.8
1.5	1×10^3	3670	764.6	11	12.5	22	21.6
2.0	4×10^4	4273	889.5	17	18.8	diverged	
2.5	4×10^6	5259	1094.4	26	28.2	diverged	

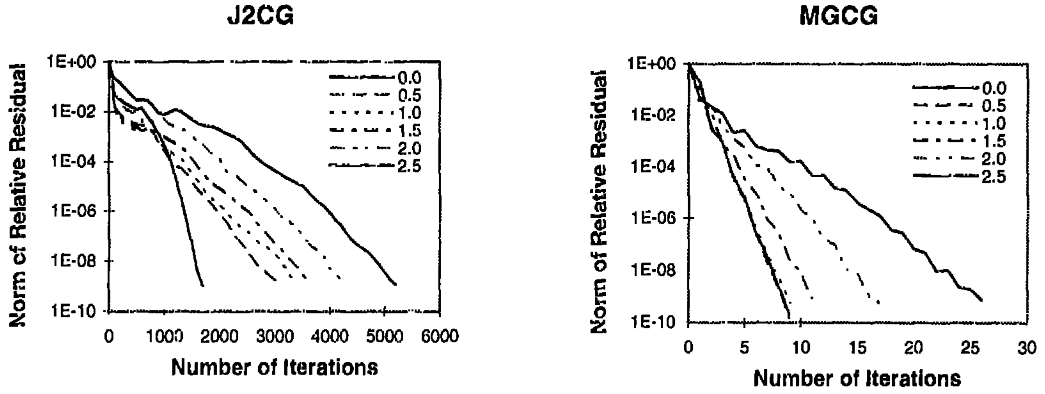


FIG. 5.1. *Convergence plots for J2CG and MGCG for several values of σ (as in Table 5.5). As σ increases, the subsurface realization becomes more heterogeneous, and the underlying matrix problem becomes more difficult.*

this increasing heterogeneity is significant, and we see that for two of the runs, it actually diverges. However, when MG is used as a preconditioner for CG (MGCG), convergence is obtained in each case. Note that the iterations for MGCG grow like the order of the variance. The convergence of J2CG is poor, as expected.

Convergence plots for J2CG and MGCG are given in Figure 5.1 for each of the values of σ in Table 5.5. Notice that the MGCG convergence curves are nearly linear and quite steep in comparison to J2CG, indicating that MGCG is making rapid and steady progress toward the solution. (The log of the 2-norm of the relative residual is plotted against the number of iterations required for convergence.)

5.5. Parallel performance on the CRAY T3D. In earlier experiments [6], we examined the parallel performance of the PARFLOW simulator and its component routines. In this section, we reprise those experiments with respect to the multigrid algorithm. Specifically, we will examine the scalability of the MGCG algorithm on the CRAY T3D massively parallel computer system. The results given here differ from those in [6] for several reasons, including compiler upgrades, algorithm enhancements, and coding improvements.

In Figures 5.2–5.3, we present scaled speedups for the matrix-vector multiplication, multigrid preconditioning, and MGCG routines on the CRAY T3D. Our machine has 256 nodes, each consisting of a 150MHz DEC Alpha processor and 64MB of memory; our AMPS message-passing library is layered on top of Cray's SHMEM library. In our experiments, each processor is given a $64 \times 64 \times 32$ subgrid, so that the total problem size on $P = p \times q \times r$ processors is $N_P = 64p \times 64q \times 32r$. In other words, we allow the total problem size to grow with P. Moreover, the shape of the problem domain is determined by the process grid topology $p \times q \times r$.

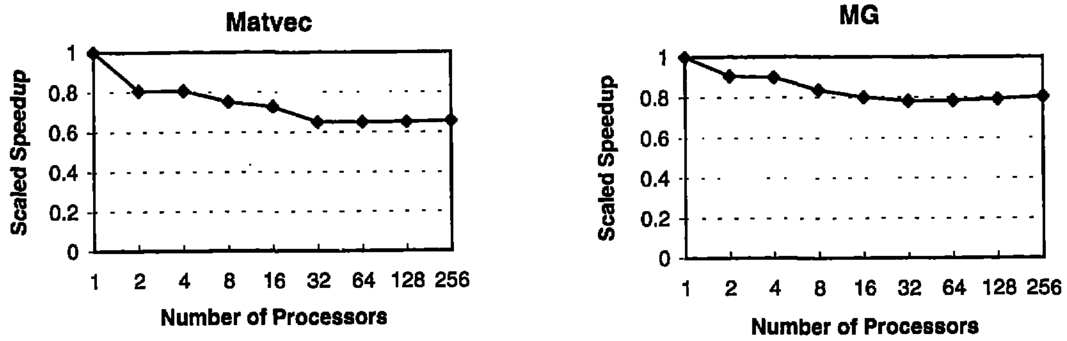


FIG. 5.2. *Scaled speedup of the ParFlow matrix-vector multiplication and multigrid preconditioning routines on the CRAY T3D.*

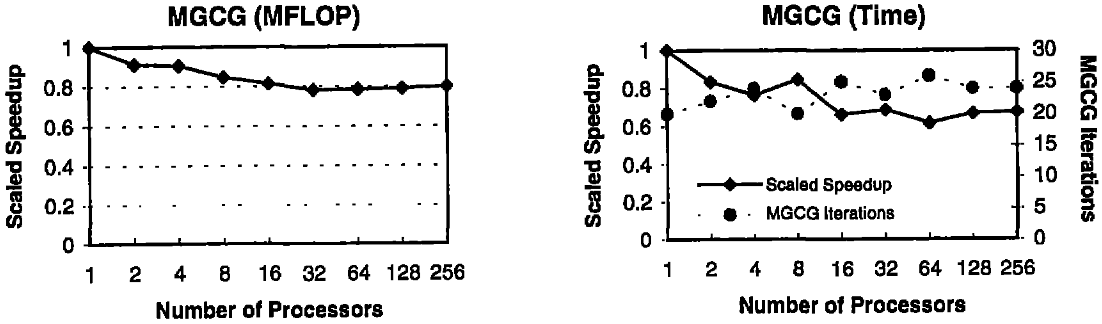


FIG. 5.3. *Scaled speedup of the ParFlow MGCG routine on the CRAY T3D. The figure on the left shows the scalability of the MGCG implementation (via MFLOP rates); the figure on right shows the scalability of the MGCG implementation* and *algorithm (via timings that include the effects of differing iteration counts).*

The point of this study is to see how well the routines make use of additional processors. Our goal is to obtain nearly flat curves (good scalability) that are near one (good scaled efficiency).

The first three graphs in Figures 5.2–5.3 measure the scalability of our *implementations* of the matvec, MG, and MGCG routines in terms of MFLOPs. Specifically, we define scaled speedup to be M_P/M_1, where M_P is the MFLOPs achieved by the operation in question on P processes. The scaled speedup graphs are all fairly flat, indicating good scalability. The MG and MGCG routines have nearly identical performance (about 80% scaled efficiency) because MGCG spends most of its time in the MG preconditioning routine. The matvec routine has lower scaled efficiency (about 65%) because it has a much higher MFLOP rate than the other routines, and so communication costs are relatively higher. (The matvec, MG, and MGCG routines averaged 2.12, 1.25, and 1.37 GFLOPs, respectively, on 256 processors.) Thus, all three routines are scalable, meaning, for example, that the time per MGCG iteration remains constant as we increase the problem size and number of processes in tandem.

In the last graph (Figure 5.3), we measure MGCG scaled speedup in terms of CPU time. That is, we define scaled speedup to be T_1/T_P, where T_P is the time required to execute the MGCG algorithm (to convergence) on P processes. Since the number of iterations required for convergence fluctuates with P, this graph measures the combined scalability of the algorithm itself *and* our implementation of it. We also plot MGCG iteration count, which varies between 20 and 26 iterations (using the C-norm stopping criterion). Notice the inverted relationship between scaled speedup and iteration count (as one would expect).

Remark: One might expect the number of MGCG iterations to increase monotonically with the size of the problem (which grows with the number of processors), but this is not the case. Recall that in our definition of scaled speedup, the computational domain is growing with P—and changing shape as we move from one process grid topology to the next. This means that the eigenstructures of the underlying matrices change from one run to the next, which accounts for the up-and-down iteration counts. We could largely eliminate this effect by keeping the domain fixed and increasing the resolution as we grow the problem size, but this would require varying the topology of the subgrid assigned to each processor. As discussed in [6], this can have a dramatic impact on node performance, causing another set of problems. (In our experiments, we used the following process grid topologies: $1 \times 1 \times 1$, $1 \times 1 \times 2$, $1 \times 2 \times 2$, $1 \times 2 \times 4$, $1 \times 4 \times 4$, $1 \times 4 \times 8$, $1 \times 8 \times 8$, $1 \times 8 \times 16$, $2 \times 8 \times 16$.)

6. Summary. This paper focuses on the numerical simulation of groundwater flow through heterogeneous porous media. The key computational challenge is the solution of a large, sparse system of linear equations for the pressure head. The size of the sites to be modeled (on the order of kilometers), and the need to resolve subsurface heterogeneities (to within a few meters), necessitates the use of efficient numerical methods and the power of massively parallel processing. In this paper, we introduce a parallel multigrid preconditioned conjugate gradient algorithm for solving these linear systems.

After defining the various components of the multigrid algorithm, and discussing its parallel implementation, we investigated its performance in a variety of numerical experiments. We considered the effects of boundary conditions, coarse grid solver strategy, increasing the grid resolution, enlarging the domain, and varying the geostatistical parameters used to define the subsurface realization. Our multigrid preconditioned conjugate gradient solver performed extremely well. For example, we were able to solve a problem with more than 8M spatial zones in under 13 seconds on a 256-processor CRAY T3D. We also demonstrated the scalability of both the algorithm *and* its implementation. This solver has been incorporated in the PARFLOW simulator and is being used to enable detailed modeling of large sites.

7. Acknowledgements. We acknowledge the valuable participation of Charles Baldwin, John Bell, William Bosl, Thomas Fogwell, Steven Smith, and Andrew Tompson in the PARFLOW project. In particular, we are indebted to Steven Smith for his fine implementation of the AMPS message-passing routines on the CRAY T3D; and we thank Andrew Tompson for his patient tutoring in the application area, especially in the use of geostatistics.

The CRAY T3D experiments described in this paper were run on the 256-processor machine located in the Center for Computational Sciences and Engineering at Lawrence Livermore National Laboratory as part of the H4P Industrial Computing Initiative funded by DOE Defense Programs.

REFERENCES

[1] R. ABABOU, D. B. MCLAUGHLIN, L. W. GELHAR, AND A. F. B. TOMPSON, *Numerical simulation of three-dimensional saturated flow in randomly heterogeneous porous media*, Transport in Porous Media, 4 (1989), pp. 549–565.

[2] R. E. ALCOUFFE, A. BRANDT, J. E. DENDY, AND J. W. PAINTER, *The multi-grid method for the diffusion equation with strongly discontinuous coefficients*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 430–454.

[3] S. F. ASHBY, W. J. BOSL, R. D. FALGOUT, S. G. SMITH, A. F. B. TOMPSON, AND T. J. WILLIAMS, *A numerical simulation of groundwater flow and contaminant transport on the CRAY T3D and C90 supercomputers*, in Proc. 34th Cray User Group Conference, Cray User Group, Inc., 1994, pp. 275–282. Held in Tours, France, October 10–14, 1994. Also available as LLNL technical report UCRL-JC-118635.

[4] S. F. ASHBY, R. D. FALGOUT, S. G. SMITH, AND T. W. FOGWELL, *Multigrid preconditioned conjugate gradients for the numerical simulation of groundwater flow on the CRAY T3D*, in Proc. ANS Interna-

tional Conference on Mathematics and Computations, Reactor Physics, and Environmental Analyses, 1995. Held in Portland, OR, April 30–May 4 1995. Refereed proceedings.

[5] S. F. ASHBY, R. D. FALGOUT, S. G. SMITH, AND A. F. B. TOMPSON, *Modeling groundwater flow on MPPs*, in Proc. 1993 Scalable Parallel Libraries Conference, A. Skjellum and D. Resse, eds., IEEE Computer Society Press, 1994, pp. 17–25. Held at Mississippi State University, October 6–8, 1993. Also available as LLNL technical report UCRL-JC-115602.

[6] ——, *The parallel performance of a groundwater flow code on the CRAY T3D*, in Proc. Seventh SIAM Conference on Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, 1995, pp. 131–136. Held in San Francisco, February 15–17, 1995. Also available as LLNL technical report UCRL-JC-118604.

[7] S. F. ASHBY, T. A. MANTEUFFEL, AND P. E. SAYLOR, *A taxonomy for conjugate gradient methods*, SIAM J. Numer. Anal., 27 (1990), pp. 1542–1568.

[8] W. L. BRIGGS, L. HART, S. F. MCCORMICK, AND D. QUINLAN, *Multigrid methods on a hypercube*, in Multigrid Methods: Theory, Applications, and Supercomputing, S. F. McCormick, ed., vol. 110 of Lecture Notes in Pure and Applied Mathematics, Marcel Dekker, New York, 1988, pp. 63–83.

[9] T. F. CHAN AND R. S. TUMINARO, *Design and implementation of parallel multigrid algorithms*, in Proceedings of the Third Copper Mountain Conference on Multigrid Methods, S. F. McCormick, ed., New York, 1987, Marcel Dekker, pp. 101–115.

[10] P. CONCUS, G. H. GOLUB, AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 309–332.

[11] J. E. DENDY, *Black box multigrid*, J. Comput. Phys., 48 (1982), pp. 366–386.

[12] ——, *Black box multigrid for nonsymmetric problems*, Appl. Math. Comput., 13 (1983), pp. 261–284.

[13] J. E. DENDY AND C. C. TAZARTES, *Grandchild of the frequency decomposition multigrid method*, SIAM J. Sci. Comput., 16 (1995), pp. 307–319.

[14] T. W. FOGWELL AND F. BRANKHAGEN, *Multigrid method for the solution of porous media multiphase flow equations*, in Nonlinear Hyperbolic Equations—Theory, Computation Methods, and Applications, vol. 24 of Notes on Numer. Fluid Mech., Vieweg, Braunschweig, 1989, pp. 139–148.

[15] G. H. GOLUB AND J. M. ORTEGA, *Scientific Computing: An Introduction with Parallel Computing*, Academic Press, San Diego, CA, 1993.

[16] W. HACKBUSCH, *The frequency decomposition multigrid method, part I: Application to anisotropic equations*, Numer. Math., 56 (1989), pp. 229–245.

[17] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–435.

[18] D. J. MAVRIPLIS AND A. JAMESON, *Multigrid solution of the Euler equations on unstructured and adaptive meshes*, in Multigrid Methods: Theory, Applications, and Supercomputing, S. F. McCormick, ed., vol. 110 of Lecture Notes in Pure and Applied Mathematics, Marcel Dekker, New York, 1988, pp. 413–429.

[19] P. D. MEYER, A. J. VALOCCHI, S. F. ASHBY, AND P. E. SAYLOR, *A numerical investigation of the conjugate gradient method as applied to three-dimensional groundwater flow problems in randomly heterogeneous porous media*, Water Resources Res., 25 (1989), pp. 1440–1446.

[20] N. H. NAIK AND J. R. ROSENDALE, *The improved robustness of multigrid elliptic solvers based on multiple semicoarsened grids*, SIAM J. Numer. Anal., 30 (1993), pp. 215–229.

[21] S. SCHAFFER, *A semi-coarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients*, SIAM J. Sci. Comput., (To appear).

[22] O. TATEBE, *The multigrid preconditioned conjugate gradient method*, in Sixth Copper Mountain Conference on Multigrid Methods, N. D. Melson, T. A. Manteuffel, and S. F. McCormick, eds., vol. CP 3224, Hampton, VA, 1993, NASA, pp. 621–634.

[23] A. F. B. TOMPSON, R. ABABOU, AND L. W. GELHAR, *Implementation of of the three-dimensional turning bands random field generator*, Water Resources Res., 25 (1989), pp. 2227–2243.

[24] A. F. B. TOMPSON, S. F. ASHBY, R. D. FALGOUT, S. G. SMITH, T. W. FOGWELL, AND G. A. LOOSMORE, *Use of high performance computing to examine the effectiveness of aquifer remediation*, in Proc. X International Conference on Computational Methods in Water Resources, A. Peters, G. Wittum, B. Herrling, U. Meissner, C. Brebbia, W. Gray, and G. Pinder, eds., vol. 2, Kluwer Academic Publishers, Dordrecht, 1994. Held in Heidelberg, Germany, July 19-22, 1994. Also available as LLNL technical report UCRL-JC-115374.